ソフトウェアで対策する場合の見分け方と見切り方

山崎 辰雄

突然の仕様変更や性能要求により，設計者は開発の過程でしばしば「新たなる課題」を突き付けられる．システムLSIや組み込みシステムの開発では，ハードウェアの設計がある程度進んでいると，ソフトウェアだけで対応しなければならないことも多い．ここでは，仕様変更などの際にソフトウェアに要求されがちな項目について整理する．（編集部）

1. 技術者の「試練」とその理由

組み込みシステムの開発時に，ハードウェアの設計が完了した後になって「この機能を追加してほしい」などと言われた経験はありませんか．または，出来上がったソフトウェアをハードウェアに載せて評価したら性能が出ないという経験をしたことはありませんか．筆者は，そのような経験をしたことが不幸にして何度かあります．各種の制約を考えると，このLSI/CPUを使わざるを得ない，足りない機能はソフトウェアで何とかしなければならない，ということもあります．今までハードウェアで担っていたある機能をそっくりソフトウェアで作れ，などと言われることもあります．そのようなとき，あなたが担当の技術者だったら，どうしますか？

「無理だ！」と思っても，できない理由を並べるだけでは仕方がありません．少なくとも「これこれがあればできます」という表現をするように心がけて，解決への道筋を探していくしかないでしょう．

● ハードウェアで対応しない理由がある

上記のような事態への対応に，ハードウェアを使わない（使えない）のにはいろいろな理由があります．ソフトウェアでもできるから，その機能をハードウェアに求めるとコストが高くつくから，ハードウェアを実装するスペースがないから，ハードウェアで実現するのは不可能だから，などです．

ただし，その機能をソフトウェアで実現するためには，何かを犠牲にしなければならない（大目に見なければならない）場合もあることを忘れてはいけません．例えば，ソフトウェアはその処理を実行する分，ほかの処理を実行できなくなってしまいます．また，ソフトウェアで実現する以上，ハードウェアよりも処理に時間がかかるかもしれません．メモリの追加や処理性能の高いCPUが必要になることもあります．ハードウェアで実現するならば適当な部品があったのに，ソフトウェアで実現したがために開発期間が長引いてしまうかもしれないのです．

● 現象は避けられないが災害は軽減できる

担当する技術者にしてみれば，このように新しく降ってわいたようにもたらされる課題は「災害」のように感じられるものです．逃れようのない災害の最たるものは地震ですが，有名な物理学者である寺田寅彦氏の言葉に，次のようなものがあります．

「いかなる震度の時にいかなる場所にいかなる程度の危険があるかということの概念がはっきりしてしまえば，無用な恐怖と狼狽の代わりに，それぞれの場合に対する臨機

KeyWord 仕様変更，機能追加，寺田寅彦，ソフトウェア，ハードウェア，システム検討，問題解決，feasibility study

図1　地震を止めることはできないが，被害は軽減できる

図2　安請け合いしなきゃよかった…

の処置ということがすぐに頭の中を占領してしまうのである」(1)

「『地震の現象』と『地震による災害』とは区別して考えなければならない．現象のほうは人間の力でどうにもならなくても『災害』のほうは注意次第でどんなにでも軽減されうる可能性があるのである」(2)

寺田氏は，これを「こわいもの」に対応するための一般的指導原理としています．これは，本稿で取り扱う組み込みシステム開発の話題にも適用できます(**図1**)．

ここでは，この原理にならって，どのような機能に関してソフトウェアでの実現を迫られることがあるのか，また，どこまではハードウェアで作らざるを得ず，どこからはソフトウェアで可能なのかを理解するところから始めたいと思います．

● 主要因は仕様変更と性能向上

そもそもどのような場合に，システムに「さらなる課題」が発生するのかを考えてみましょう．

- 機能追加/変更のための製品仕様の変更．競争の厳しい製品分野では，他社が新機能を出してきたとき対抗せざるを得ないことがある．
- コストダウンのためのハードウェア仕様の変更．「今までのチップより，このチップの方が3割安いから切り替えることにした．いろいろ変わるけど何とかしてね」と言われる場合．
- 設計ミスが後から発覚した場合(これは言語道断！)．
- 既存の設計では十分な性能が出ない場合．「現状のソフトウェアではハードウェアの性能を生かし切れていない」と判断されると，作り直しを要求される．

表1　○○が足りない

静的なもの	メモリ容量，入力ポート，出力ポート，A-Dコンバータ，D-Aコンバータ，カウンタまたはタイマ，シリアル・ポート，専用機能(エンコーダ，デコーダ，フィルタなど)
動的なもの	処理速度(転送レートなど)

これらの課題を「…が足りない」という観点でとらえ，静的なものと動的なものに分けてみます(**表1**)．このうちソフトウェアで対応できるものを考えると，主に，採用するCPUの機能不足に起因するものが多いようです．

● ソフトウェアで対応できるかどうかを検討する

ハードウェアとソフトウェアのいずれで対応するにせよ，そもそも，その要求が実現できるかどうかを検証する必要があります．次に，ハードウェアとソフトウェアのいずれで実現するのかを検討します．「ソフトでやってよ」という依頼を受けても，安請け合いせず，十分に検討しなければなりません．さんざん待たせた揚げ句，「やはりソフトウェアではできませんでした」という回答は，聞き入れてもらえないのです(**図2**)．

1) ハードウェアで実現する場合

一般的に，ハードウェアの変更で対応するのは，ソフトウェアではどうしようもないときや，比較的容易にハードウェアを変更できるときに限られます．

この方法は，ハードウェアの構成変更を伴いますが，場合によっては部品の置き換えや削除で，より優れたシステムを実現できることもあります．

中長期的には，既存の部品よりもその応用製品に適した部品を作ってしまうという選択肢もあります．ソフトウェア技術者に十分な経験と知識があれば，自分たちに使い勝手の良いものを構想できます．今までよりも優れたアーキ

1
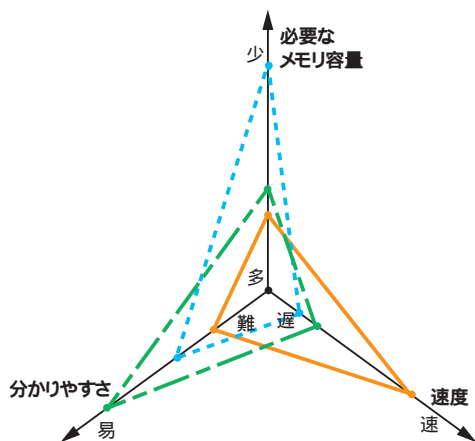
図3　メモリと速度，分かりやすさのトレードオフ
何かを得るためには，何かを諦めなくてはならない．

テクチャのLSIを作ることもできるのです．うまくすれば，長期にわたって相互利益を図ることができます．

2）ソフトウェアで実現する場合

ソフトウェアで何とかなりそうな可能性があれば，肝心なところだけを作って評価し，ほかの機能に影響がないかを確認します（いわゆるfeasibility study．日本語に訳すと「実現可能性検討」か）．

ソフトウェアで実現する場合，「メモリ，速度，分かりやすさ」のどれかをあきらめて，どれかを得ることになります（図3）．例えば，「メモリ効率を犠牲にして速度を得る」ための代表的な方法として，処理をループにせず，命令を並べてしまうやり方があります．処理速度を上げるためにソース・コードを最適化した結果，「分かりやすさ」が失われてしまうこともあります．また，機能をソフトウェアで実装して制御回路を取り除くことは，「メモリ容量と処理速度をあきらめる」ことに該当します．

いずれの作業にせよ，作業過程で「知恵と勇気」が合い言葉になることはあっても，「努力と根性」に頼ることはできれば避けたいものです．

コードの最適化アルゴリズムや手法の参考資料として，「珠玉のプログラミング」[3]と「ハッカーのたのしみ」[4]の2冊を挙げておきます．

● ソフトウェアで実現する際の考え方，手順

「必要は発明の母」と言います．どうしても実現しなければならない，という切迫感が創造性にとって大切です．ただし，切迫感だけではだめで，知識が必要です．さらに知識だけでもだめで，知識を実践できる「知恵」が必要です．

とっかかりとして，まず問題（要求）を分析します．要求を細分化して条件を探し出します．理解できたことを除いていって，未知のことに取り組みます．問題が複雑であれば，等価でありながらより容易な問題に置き換えられないかを検討します[注1]．

次に，納得のいく答えが得られるまで設計を何度も捨てて新しく試み直します．良い解決策は「どうしてもそうなってしまう」ものに落ち着くものです．優れた設計者は，窮屈な条件の中で，見栄え良く使い勝手の優れた設計を考え出すことができます．

問題を解くためには，問題を表現するのに適したソフトウェア言語を選ぶ必要があります．なぜなら，言語が考え方の枠組みを定めるからです．できるだけ「強い言語」[注2]を選ぶべきですが，ハードウェアの性能を極限まで使い切るためには，部分的に，人間に不親切なアセンブリ言語を使わなければならないこともあるかもしれません．

問題をより良く解こうと思ったら，関連した問題を解こうとしている才能の集まる場所に行く必要もあるでしょう．昔なら米国Xerox社のPalo Alto研究所，最近だと米国IDEO社や米国Google社が問題解決に最高の場所とされています．

どうにも名案を思いつかないといったときには，発想法に頼るのもよいかもしれません．一例として，TRIZ（Teoriya Resheniya Izobretatelskikh Zadatch；発明的な問題解決の理論）[5]というプロセス化された発想法を紹介しておきます．手順に沿って進めていくことにより，問題を正しく把握して，何らかの解が得られるという発想法です．

● 解決策の評価にはセンスが必要

複数の人から解決策が提案されたとき，すべてを「人それぞれのやり方がある」と無条件に受け入れては，優れたものを実現できません．

問題は，その性質により2通りに分けられます．一つは，

注1：例えば，99×99を計算するとき，99×（100－1）＝99×100－99と考えれば暗算できる．
注2：ここでは，言語Aと言語Bを比較して，言語Aでは表現できない概念を言語Bでは表現できる場合に，「言語Bは言語Aより強い言語である」と言っている．強い言語の方がプログラムの生産性は高くなるが，コンピュータで実行した場合，必ずしも効率良く実行できるとは限らない．しかし，昔と比較してハードウェアが強力になる一方，記述しなければならないソフトウェアの量が増え，内容も複雑になってきていることにより，実行効率よりも生産性向上を重視する必要性が増している．

誰でも解決できるような易しい問題です．これは容易に片付きます．

もう一つは，とても困難な問題であり，専門家にとってはできないことが常識になっているような問題です．このような場合，常識的な考えで評価する「会議」は，確実に発想の芽をつぶしてしまいます．常識で解決できなければ非常識に考える必要がありますが，どのように検討すればよいのでしょうか．

答えは，「センスのある人に判断をゆだねる」です．

技術とセンスには個人差があります．技術は没個性なもので，手順を踏んで練習すれば，誰もが一定の水準に達することができます．一方，センスは属人的なものです．センスのある人が設計した物は，単に問題を解決するだけでなく，エレガントな解決策になります．

良い設計は単純です．単純にするためには，本当の問題と向き合い，装飾ではなく実体を見せなければなりません．不要なものを取り除き，単純で簡素にするためには，かなり努力しなければならないでしょう．

● 変更への対応を「予防」する

いざ変更依頼が来てから右往左往するより，もっと良い方法があります．このような「ご相談」が入ってくる根本に対して，目を光らせておく（気を配っておく）ことです（図4）．ハードウェア技術者が何をしているのか，企画部門が何を計画しているのかなどを把握しておき，それに対して自分たちはどのような手が打てるのかということに考えを巡らせましょう．

筆者としては，できるだけ柔軟な構造でソフトウェアを作っておくことをお勧めします．あらかじめハードウェアが頻繁に変わる個所と変わりにくい個所を分析します．そして，変わりにくい個所に対応したソフトウェアを基盤とし，変わる個所をライブラリとして差し替えられるようにするのです．

分かりやすい例として，ハードウェア開発における試作機について考えてみましょう．例えば携帯電話機であれば，マイクやスピーカ，電池/電源回りは変わりにくく，液晶やCPU回り，通信機能などは製品ファミリによって変わります．試作基板を作るときには変わらない個所を土台とし，変わる個所を差し替えられるモジュール構造にして使い回しを図ります．ソフトウェア開発も同様です．

図4 「ご相談」の発生源に目を光らせておく

2. 融合しつつあるハードとソフト

「ハードウェア」，「ソフトウェア」と一言で言っても，どこからがハードウェアでどこからがソフトウェアなのかを区別するのは年々難しくなっています．

● ハードウェアとソフトウェアの境界

これまでは，何のためにハードウェアとソフトウェアを分けてきたのでしょうか．筆者は，必要な専門知識の違いだと考えます．

学生時代に何を学んだのかは関係なく，まともな文書が書けるならソフトウェア技術者になることができます．しかし，ハードウェア技術者には専門知識が必須でした．具体的には，電子工学科や電気工学科，高等専門学校などの専門教育を受けて身に付ける，電気や部品についての知識が必要でした．

それが，FPGA（field programmable gate avray）やPLD（programmable logic device）といったプログラム[注3]可能なハードウェアが使えるようになり，担当個所によっては，電気的な知識がなくても論理的な思考だけで手軽に作ることができるようになってきました．このとき，ハードウェアはFPGAという部品までで，それをプログラムして所望の機能を実現することをプログラミングと呼ぶことを考えると，FPGAを使った設計作業は「ソフトウェア開発」と言えるのではないでしょうか．

FPGAをプログラムするときに限れば，どのような道具

注3：ここでいう「プログラム」とは，そのハードウェアで実現する機能のカスタマイズを意味する．

1

表2
ハードウェアとソフトウェアの比較

	ハードウェア	ソフトウェア
考え方	・物理法則に支配される ・適切な構成要素を選択し，構成要素間の物理的な接続関係を考える	・論理に支配される ・アルゴリズムが大きな要素になる．手続きを並べていく（利用する言語にもよる） ・処理時間はプロセッサのクロック次第
設計時にイメージするもの	・電気の操作（信号とそのタイミング） ・機構の操作（スイッチなどの可動部品）	・データ（情報）の操作
最終的な表現	・構成要素とその接続の表現が主体 ・タイミング図や回路図，ブロック図などで表す	・状態や制御の流れ，振る舞いが主体 ・状態遷移図やフローチャート，シーケンス図などで表す．最終成果物はソフトウェア・コード

（ツール）を使ってプログラムする内容を記述するかによって，ハードウェア技術者にしかできなかったり，ソフトウェア技術者の方が楽にこなしてしまうものもあります．何をハードウェアと呼び，何をソフトウェアと呼ぶのかは，作るときの考え方に求めた方がよい時代になったと思います．

● ハードウェア的な考え方とソフトウェア的な考え方

1946年に初公開されたコンピュータ「ENIAC（Electronic Numerical Integrator and Computer）」におけるプログラミングとは，配線を付け替えることでした．さて，これはハードウェアなのでしょうか，ソフトウェアなのでしょうか．扱う「物」で区別すると，これは結線論理（wired logic）ですから，答えは「ハードウェア」となります．「考え方」で区別すると，「ソフトウェア」だと言えます．ハードウェアとソフトウェアの考え方などの違いを**表2**に示します．

ただし，システムとして検討するときは，まずはハードウェアとソフトウェアのどちらで実現するのかは考えずに，最終的なシステム全体をイメージする必要があります．少なくとも，全体を構成するアーキテクトの頭には，そのようにイメージされていなければなりません．

次に，利用可能なハードウェアなどを考慮しながら，どの部分をハードウェアに担当させるのかを割り出します．それから，ソフトウェアで実現しなければならない個所を検討します．それぞれの詳細を規定する段階になって初めて，それぞれのイメージで考え始めます（**表2**）．

製品を設計するとき，ハードウェア的な考え方とソフトウェア的な考え方を両方使うことにより，「物作り」の醍醐味を存分に味わえることでしょう．

● 最適化したら「複合機能部品」として取り扱う

組み込み製品開発では，ハードウェアとソフトウェアの間に橋を架けることが頻繁に起こります．最適化を進めていくとソフトウェアとハードウェアが渾然一体となり，どこからどこまでがどちらなのか見分けがつかなくなります．そして，個別機能部品として切り出せなくなります．こうなると「再利用性に劣る」ともみなされますが，再利用時には複合機能部品として取り扱うことになります．

参考・引用*文献

(1)* 寺田寅彦；家庭の人へ，寺田寅彦全集 第二巻，岩波書店，1997年1月．
http://www.aozora.gr.jp/cards/000042/files/42161_18162.html

(2)* 寺田寅彦；災難雑考，寺田寅彦随筆集 第五巻，岩波文庫，1948年11月．
http://www.aozora.gr.jp/cards/000042/files/2500_13848.html

(3) Jon Bentley著，小林健一郎訳；珠玉のプログラミング，ピアソンエデュケーション，2000年10月．

(4) Henry S. Warren，Jr.著，滝沢 徹ほか訳；ハッカーのたのしみ，エスアイビーアクセス，2004年9月．

(5) 中川 徹；TRIZホームページ，http://www.osaka-gu.ac.jp/php/nakagawa/TRIZ/

やまさき・たつお
ニュアンス コミュニケーションズ ジャパン（株）

<筆者プロフィール>
山崎辰雄．興味の向くままハードウェアとソフトウェアを楽しみ，IT業界，組み込み業界を渡り歩く「流しの技術者」．最近は音声合成，音声認識を取り扱う．組込みソフトウェア管理者・技術者育成研究会（SESSAME）のメンバ．